Homework 11 Introduction to Big Data Systems

Christoffer Brevik

November 27, 2024

1 Code Overview

To solve this weeks assignment I have implemented a Python Spark Streaming program computing the Top-k frequent words from files streamed into an HDFS directory. The job processes incoming files, which are sent every minute, combining their results with historical counts, and outputs top-k frequent words from the combined results.

1.1 Dependencies

For this assignment I did not add any dependencies not already in the computational node, and I am able to run it at time of delivery.

1.2 How to Run the Code

To execute the program from the computational node:

1. Open a terminal in root and move to the generator folder with `cd generator`.
2. With the terminal, now run `bash generator.sh`. This script will now start streaming the files every minutes, and I will refer to this terminal as *Terminal 1* moving forwards.
3. Open a new Terminal, *Terminal 2*, in root and move to the my codes folder with `cd my_python code`.
4. Now start my script by running `bash submit.sh`
5. Allow the program to process up to five files.
6. For each iteration, the results are displayed in the console.
7. Stop the processes in *Terminal 1* and *Terminal 2*, and close the terminals

2 Code Implementation

The program uses PySpark Streaming to process incoming files and compute word frequencies. I've seperated my code into three sections, so they can be run from a Jupyter library and are easier to explain:

2.1 Streaming Context Setup

- A `SparkContext` and `StreamingContext` are initialized with a batch interval of 60 seconds.
- The input source is a text stream from the specified HDFS directory.
- The global values for my code are defined. **A global dictionary** containing all words and their counts, used to maintain cumulative word counts across batches. **The current File number**, used to order the data in the terminal and for stopping the program in time, and how many files we should parse in total

2.2 Core Functions

My program contains two core functions used when calculating the top-k words.

2.2.1 update_count

This function merges the word counts from the current RDD with historical counts stored in `word_counts`. It handles both initialization and incremental updates of the word-count. This means it works for both the first and all following files.

2.2.2 process_rdd

This function processes each RDD to:

- Extract words from lines and compute word counts.
- Update the global word counts using `update_count`.
- Sort the updated word counts and select the top-k frequent words.
- Saves the result to a file.
- Also prints the same results to the console
- If the maximum numbers of files have been reached, we stop processing.

2.3 Running the code

For every RDD that is picked up, we run the *process_rdd*, which runs the *update_count*. As explained, these functions stop themselves in time. After processing five files, they stop both the streaming job and the Spark context gracefully.

3 Results

3.1 Console Output

For each processed file, the program outputs:

- A header indicating the file number.
- The top-100 frequent words in descending order of frequency, along with their counts.

These are both saved as a file and printed to the console. I will now show the results of me running my algorithm:

3.1.1 File 1 Output

Below is a sample of the top-10 frequent words for File 1. Full results (top-100) are available in the console output or saved files.

Word	Count
m36	7
o2	6
p3	6
m29	6
n33	6
r20	6
o31	6
p10	6
n32	6
p14	6

Table 1: Top-10 frequent words for File 1.

3.1.2 File 5 Output

The cumulative top-10 frequent words after processing all five files are shown below. Full results (top-100) are available in the console output or saved files.

Word	Count
m36	26
k30	20
m4	19
n9	19
m8	19
m19	18
o12	18
l22	18
o26	18
n16	17

Table 2: Top-10 cumulative frequent words after processing 5 files.